Recurrent and Masking-based Autoregressive Models

Lecture 3

18-789

Administrative

- Please sign up for student presentation! We will assign randomly if you don’t sign up by tonight.
- Today
  - RNNs
  - LSTMs
  - MADE
  - PixelRNN

Recap: N-gram language models

Left-to-right language models

$P(x_1, x_2 \cdots x_t) = P(x_t | x_1 \cdots x_{t-1}) \cdot P(x_1)$

$P(x_{t+1} | x_1 \cdots x_t)$

What do pigs __

Main Problem?

- Problematic even for unigram
  - Poor generalization
- Inability to use long context

1. Get probability distribution

Softmax($x^T W$)

2. Sample from probability distribution

Eat 0.5
like 0.3
wear 0.01
…

Eat 0.5
like 0.3
wear 0.01
…

$P(x_t | x_{t-1}, x_{t-2} \cdots x_{t-n})$

Indep-ut Assumption.

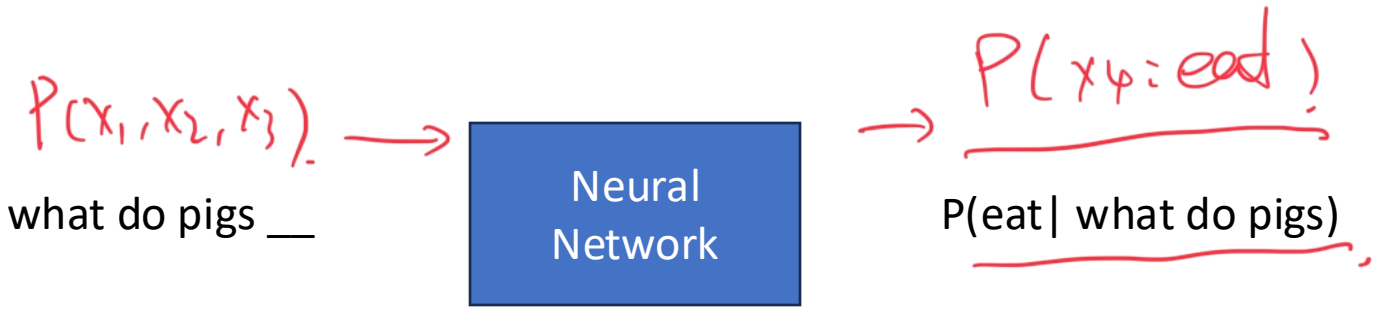
Recap: Neural Autoregressive Models

How to compute the conditional probabilities?

- Train a neural network to predict them

1. Process context → RNNs, Masking-based Models,

2. Generate a probability distribution for the next token



Recap: How Can We Choose Models?

Two variables: y_1, y_2
Model: $p(y_1, y_2) = p(y_1)p(y_2|x_1)$

$P(y_1) \rightarrow$ histogram

$P(y_2|x_1) \rightarrow$ MLP. $\rightarrow$ better than d^2.

Does it extend to high dimensions?

- For d-dim data, O(d) parameters
  - But what about d is really large?
- Limited generalization
  - No info shared among different conditions

Fig generated by Chat-GPT 4o

Recurrent neural networks

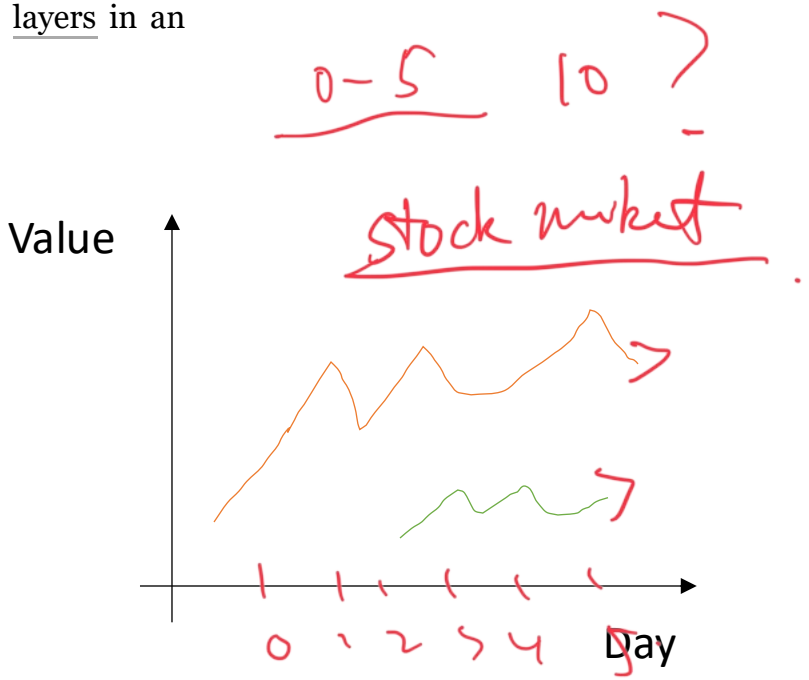
History

The Ising model (1925) by Wilhelm Lenz[9] and Ernst Ising[10][11] was the first RNN architecture that did not learn. Shun'ichi Amari made it adaptive in 1972.[12][13] This was also called the Hopfield network (1982). See also David Rumelhart's work in 1986.[14] In 1993, a neural history compressor system solved a "Very Deep Learning" task that required more than 1000 subsequent layers in an RNN unfolded in time.[15]

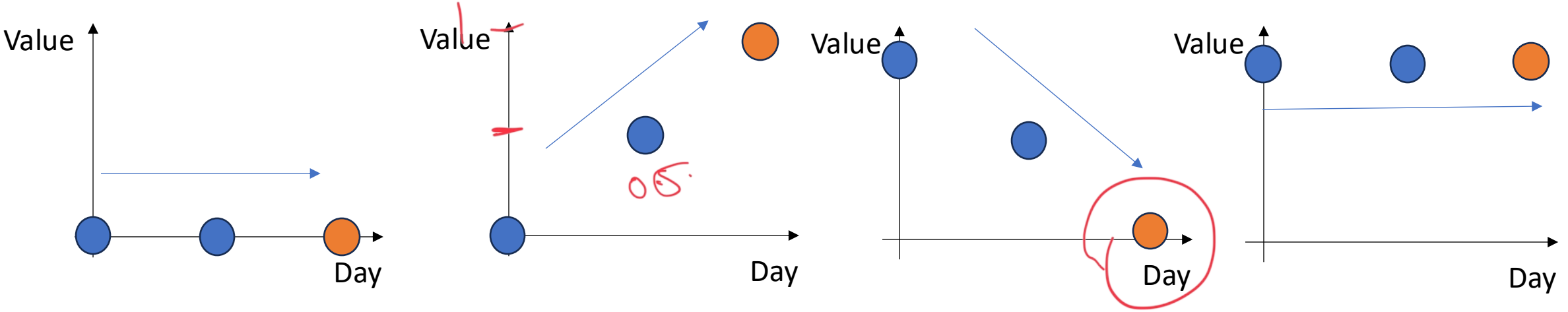
Why RNNs? A toy example:

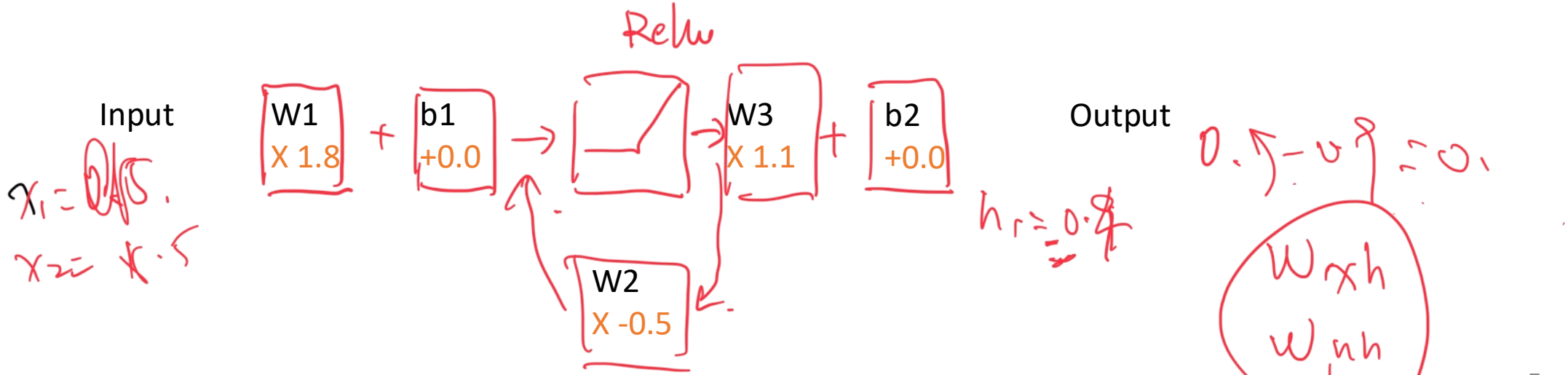
Problem with Feed Forward Networks?

- Maximum number of sequence length
- Processing at the beginning of a sequence does not generalize to the end



Recurrent neural networks





Recurrent neural networks

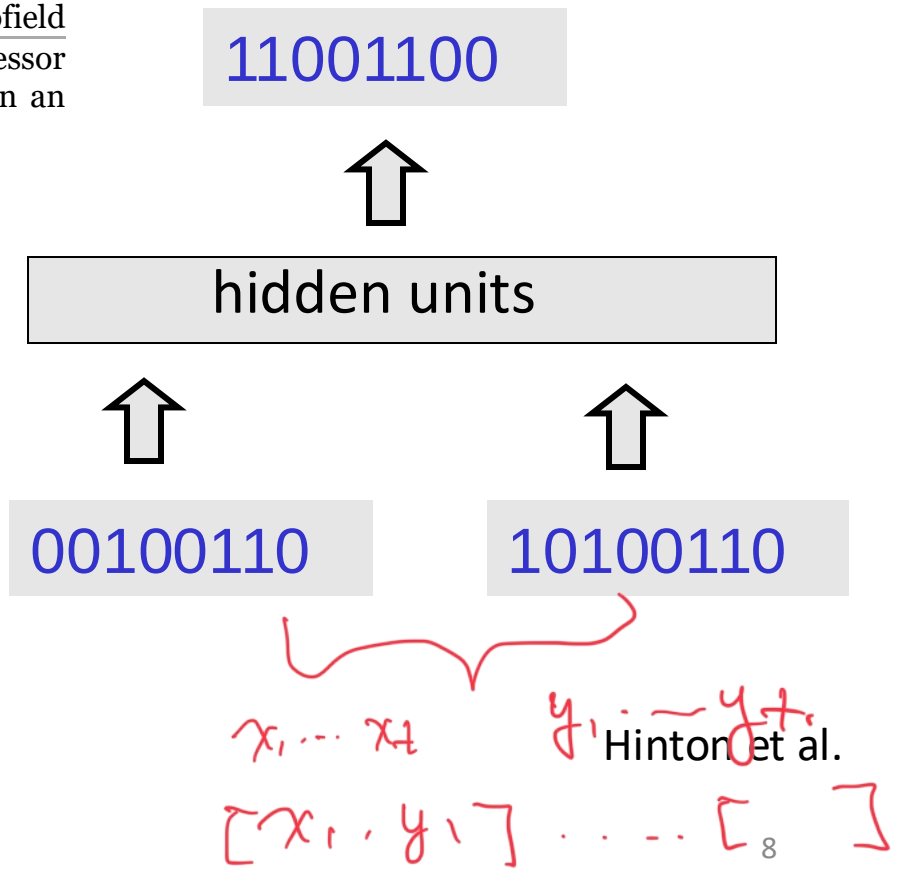
History

The Ising model (1925) by Wilhelm Lenz[9] and Ernst Ising[10][11] was the first RNN architecture that did not learn. Shun'ichi Amari made it adaptive in 1972.[12][13] This was also called the Hopfield network (1982). See also David Rumelhart's work in 1986.[14] In 1993, a neural history compressor system solved a "Very Deep Learning" task that required more than 1000 subsequent layers in an RNN unfolded in time.[15]

Why RNNs? A toy example:

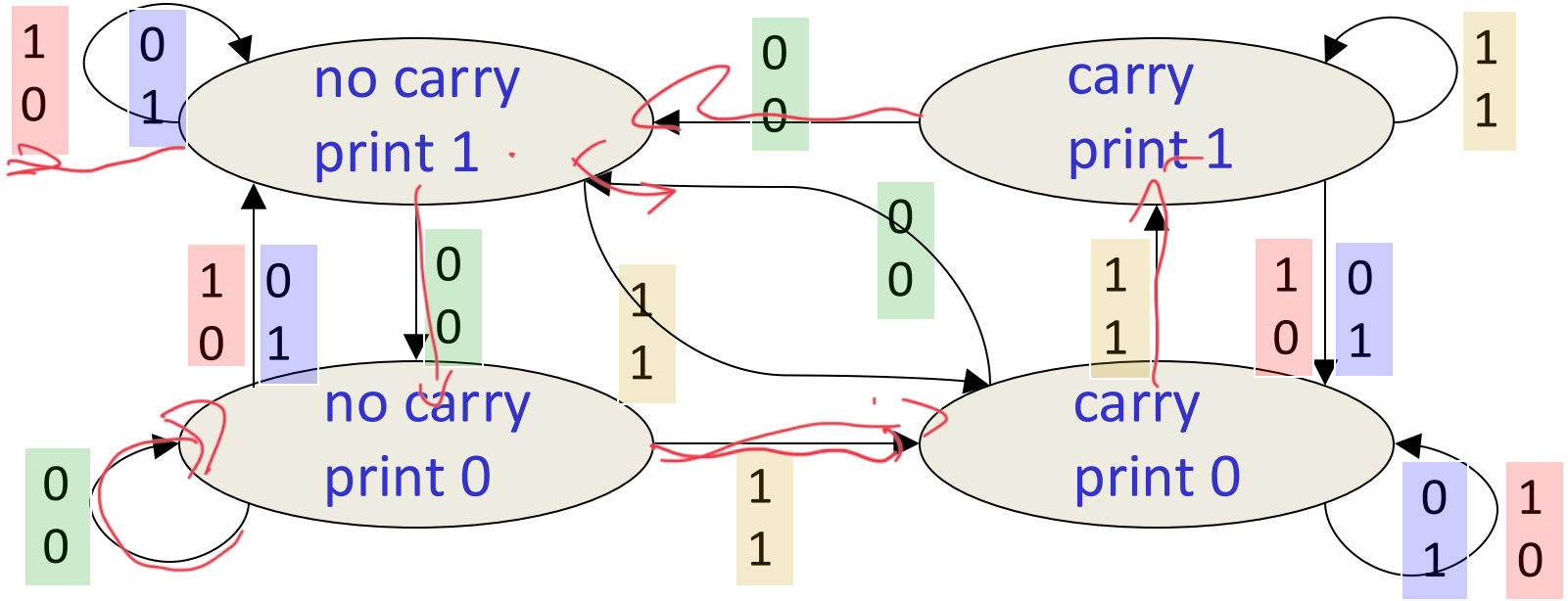
Problem with Feed Forward Networks?

- Maximum number of sequence length
- Processing at the beginning of a sequence does not generalize to the end

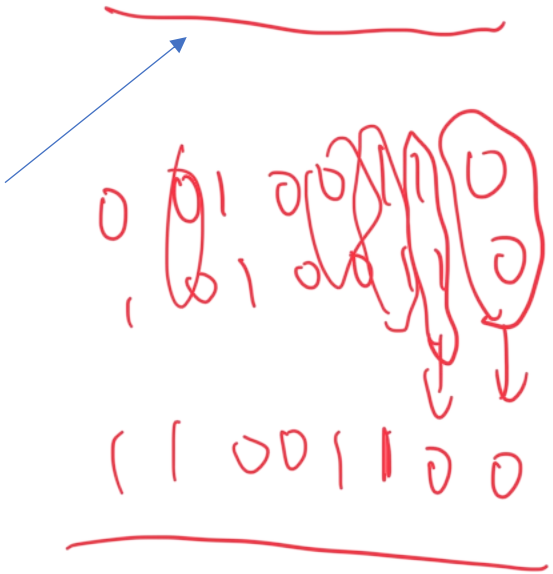


Recurrent neural networks

The algorithm for binary addition:



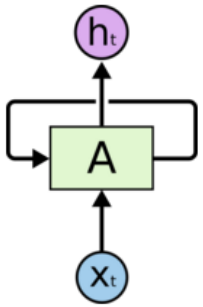
Finite state automaton



Hinton et al.

Recurrent neural networks

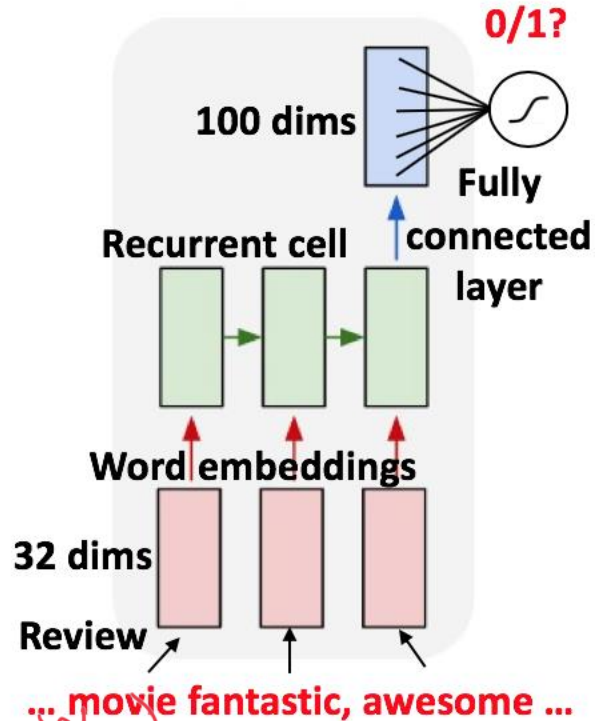
- Recurrent Neural Networks are networks with loops, allowing information to persist.



Recurrent Neural Networks have loops.

In the above diagram, a chunk of neural network, $A = f_W$, looks at some input x_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next.

Output is to predict a vector h_t, where $output\ y_t = \varphi(h_t)$ at some time steps (t)



$$h_t = f_W(h_{t-1}, x_t)$$

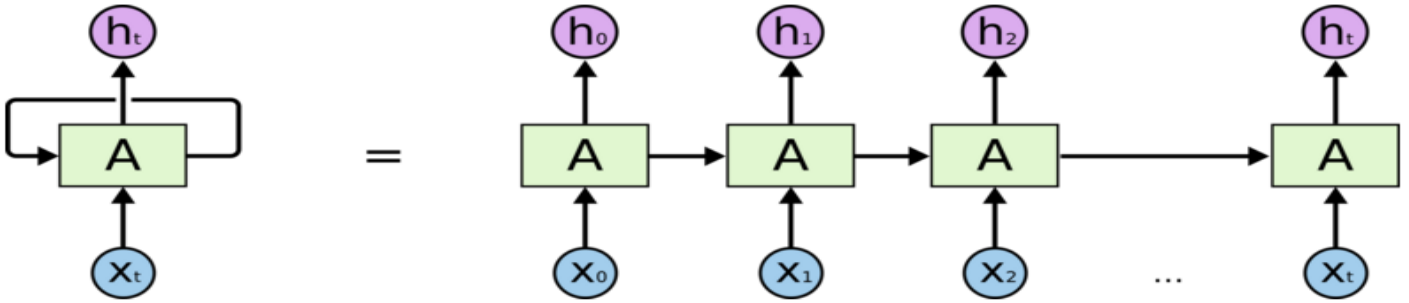
new state — old state

function with parameter W

Input vector at some time step

Recurrent neural networks

- Unrolling RNN

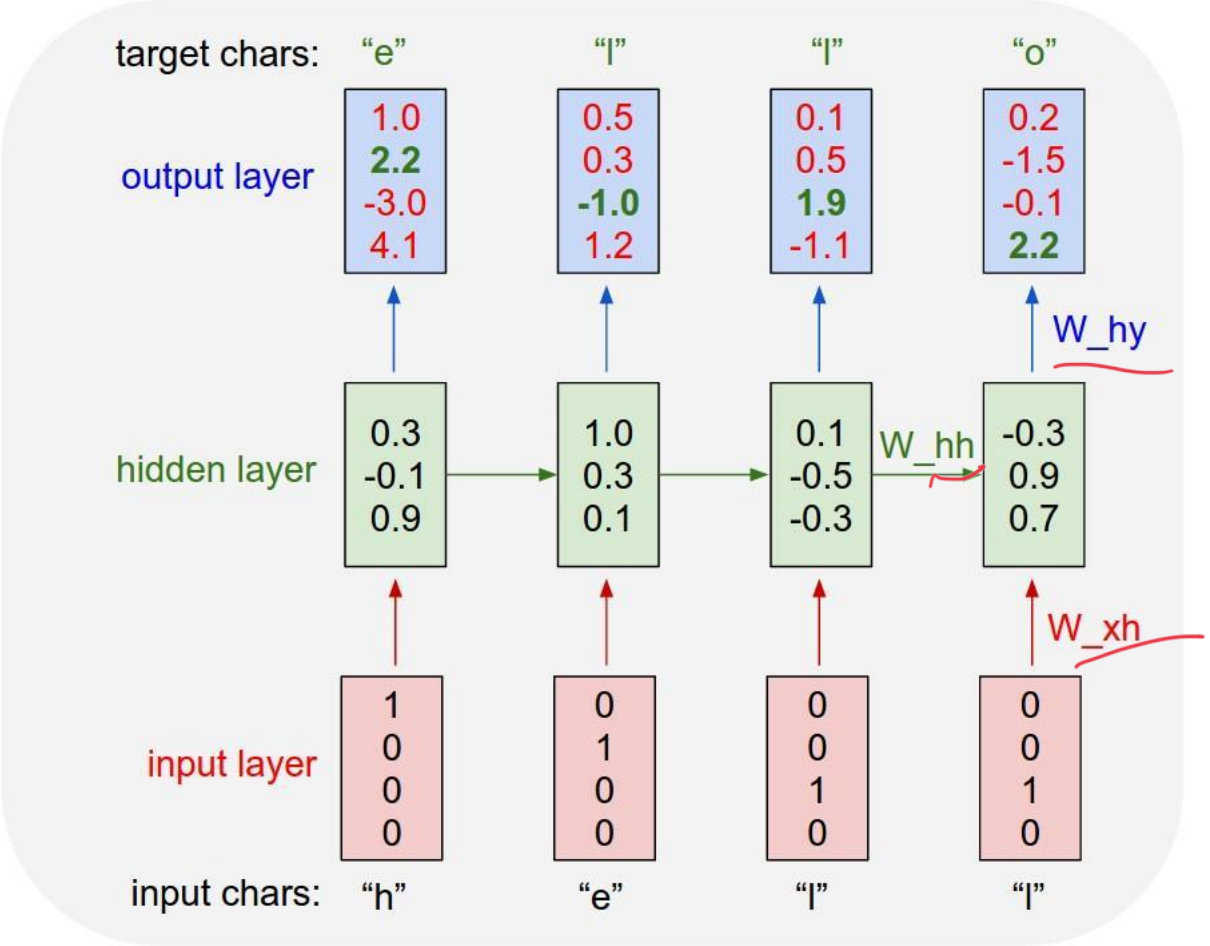


An unrolled recurrent neural network.

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. The diagram above shows what happens if we **unroll the loop**.

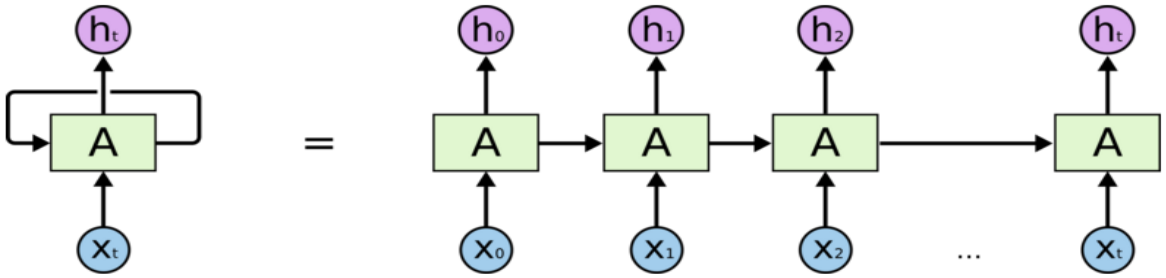
Recurrent neural networks

- Example



Recurrent neural networks

For binary addition task



An unrolled recurrent neural network.

0 0 1 1 0 1 0 0

0 1 0 0 1 1 0 1

1 0 0 0 0 0 0 1

2dim

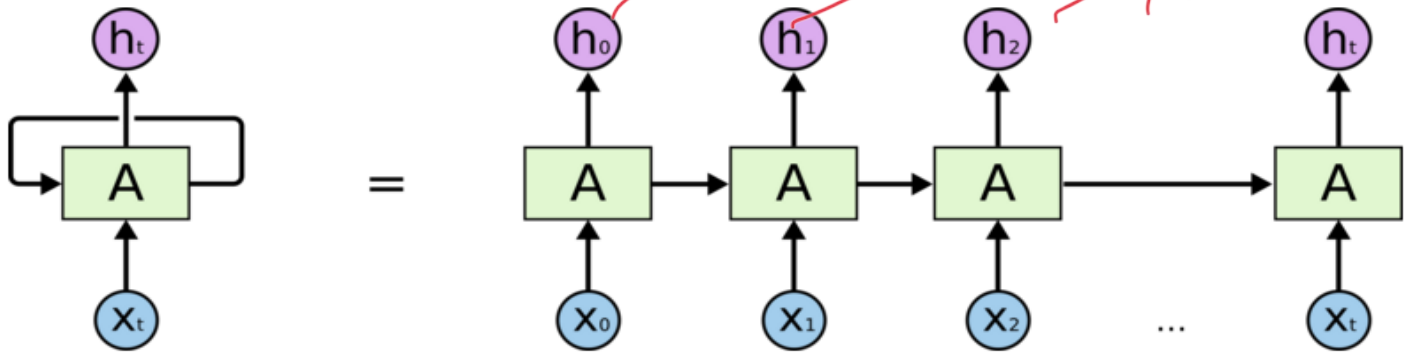
time

- The weight from input layer to hidden which the shape is (2,16)
- The weight from hidden layer to output, which the shape is (16,1)
- The weight from hidden layer to hidden layer, which the shape is (16,16)

2 x 16

12.

Back propagation through time



An unrolled recurrent neural network.

$L(\theta)$

$h_0 \to h_1 \xrightarrow{w} h_2 \xrightarrow{w} h_3$

$$\frac{\partial L(\theta)}{\partial w} = \frac{\partial L(\theta)}{\partial h_3}\frac{\partial h_3}{\partial w}$$

$h_3 = G(wh_2 + b)$

$$\frac{\partial L}{\partial h_3}\left(\sum \frac{h_3}{h_k}\frac{\partial h_k}{\partial w_{hh}}\right)$$

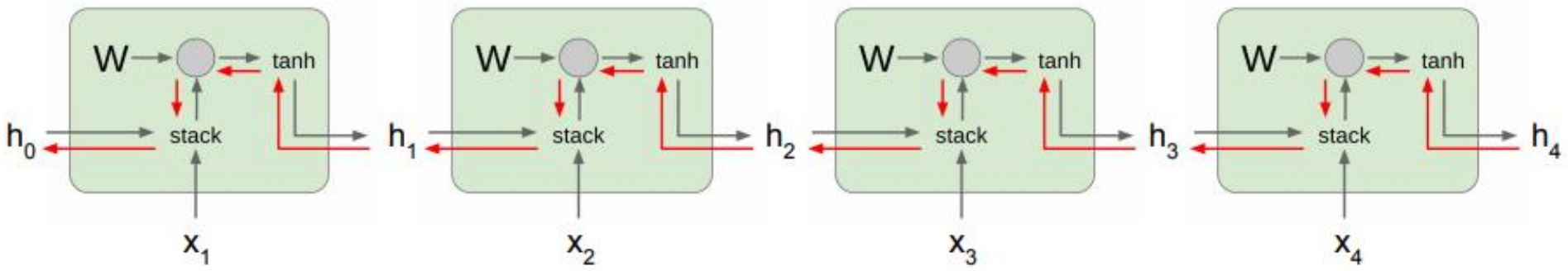
$$\frac{\partial h_3}{\partial w} = \frac{\partial h_3}{\partial w} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial w} = \frac{\partial h_3}{\partial w_1} + \frac{\partial h_3}{\partial h_2}\left[\frac{\partial h_2}{\partial w} + \frac{\partial h_2}{\partial h_1}\right]$$

Recurrent Neural Networks

The recurrent structure of RNNs enables the following characteristics:

- Specialized for processing a sequence of values $x^{(1)}, \ldots, x^{(\tau)}$
  - Each value $x^{(i)}$ is processed with the **same network *A* that preserves past information**
- Can scale to much **longer sequences** than would be practical for networks without a recurrent structure
  - Reusing network ***A*** reduces the required amount of parameters in the network
- Can process **variable-length sequences**
  - The network complexity does not vary when the input length change
- However, vanilla RNNs suffer from the training difficulty due to **exploding and vanishing gradients**.

Exploding and Vanishing Gradients



Input → W1 X 1.8 → b1 +0.0 → W3 X 1.1 → b2 +0.0 → Output

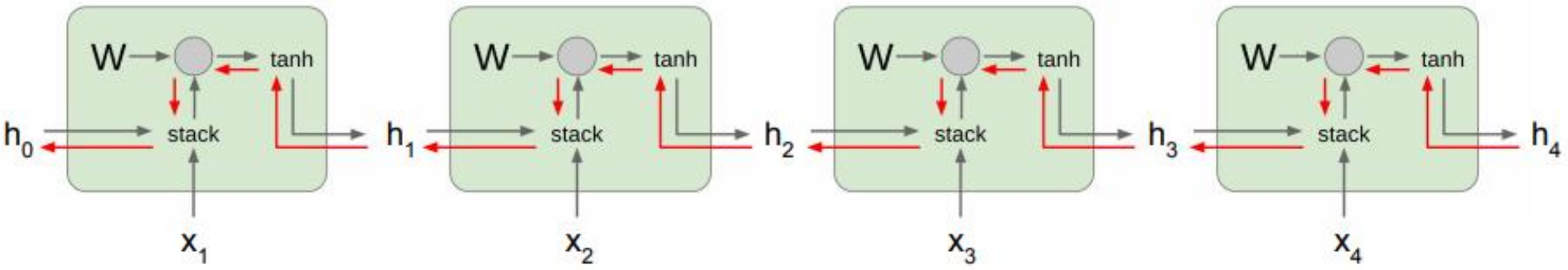
W2 X -0.5

$\frac{\partial h_3}{\partial h_1}$ W^3

$\frac{dx_i}{dw_{hh}} \frac{d(\quad)}{dw_{hh}}$ ipt $\propto w_c x_c$

* Refer to Bengio et al. (1994) or Goodfellow et al. (2016) for a complete derivation

Exploding and Vanishing Gradients



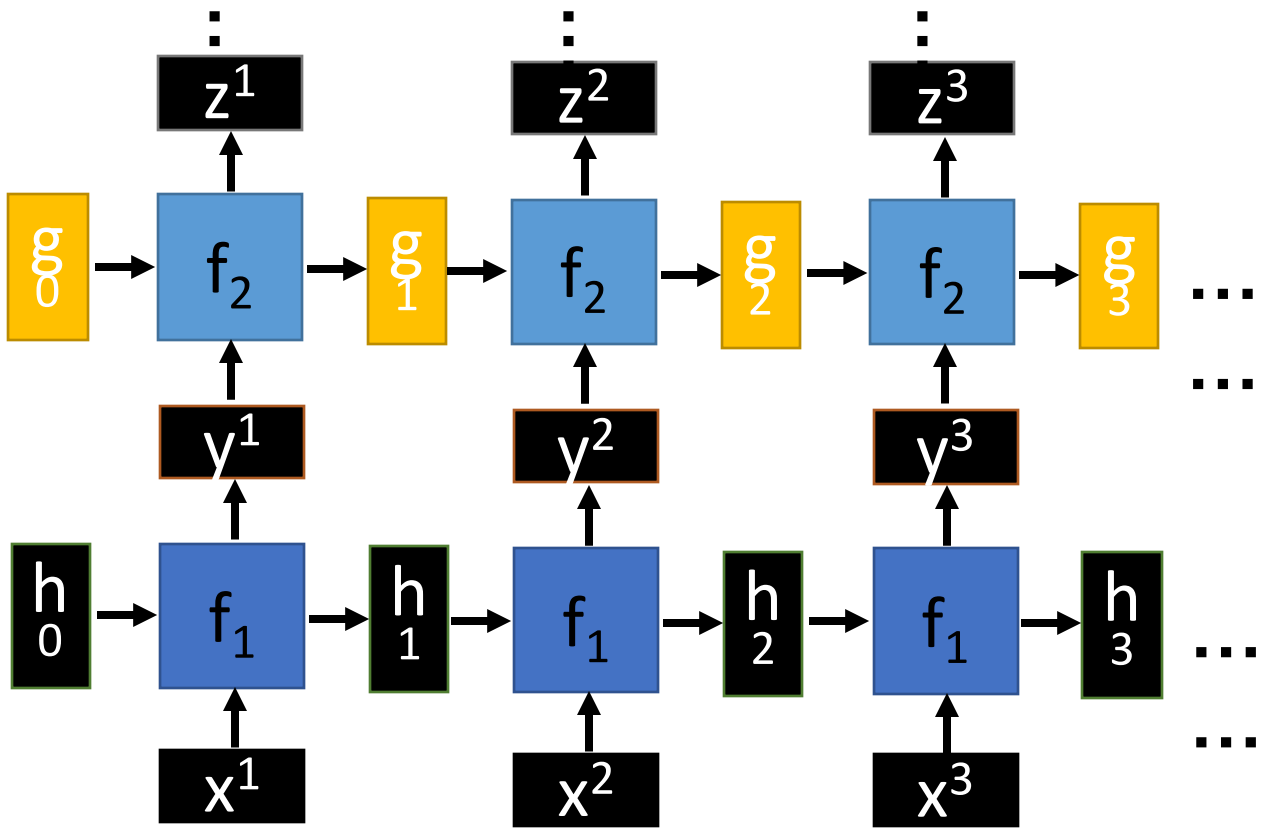
In vanilla RNNs, computing this gradient involves many factors of W_{hh} (and repeated $\tanh$)*. If we decompose the singular values of the gradient multiplication matrix,

- Largest singular value > 1 → **Exploding gradients**
  - Slight error in the late time steps causes drastic updates in the early time steps → Unstable learning
- Largest singular value < 1 → **Vanishing gradients**
  - Gradients passed to the early time steps is close to 0. → Uninformed correction

* Refer to Bengio et al. (1994) or Goodfellow et al. (2016) for a complete derivation

Deep RNNs

$h', y = f_1(h, x),\ g', z = f_2(g, y)$



Long Short-Term Memory (LSTM)

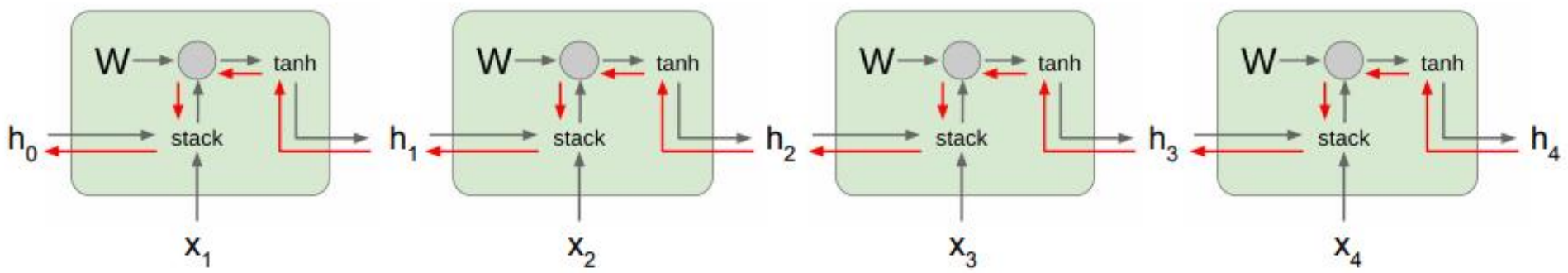
Communicated by Ronald Williams

Long Short-Term Memory

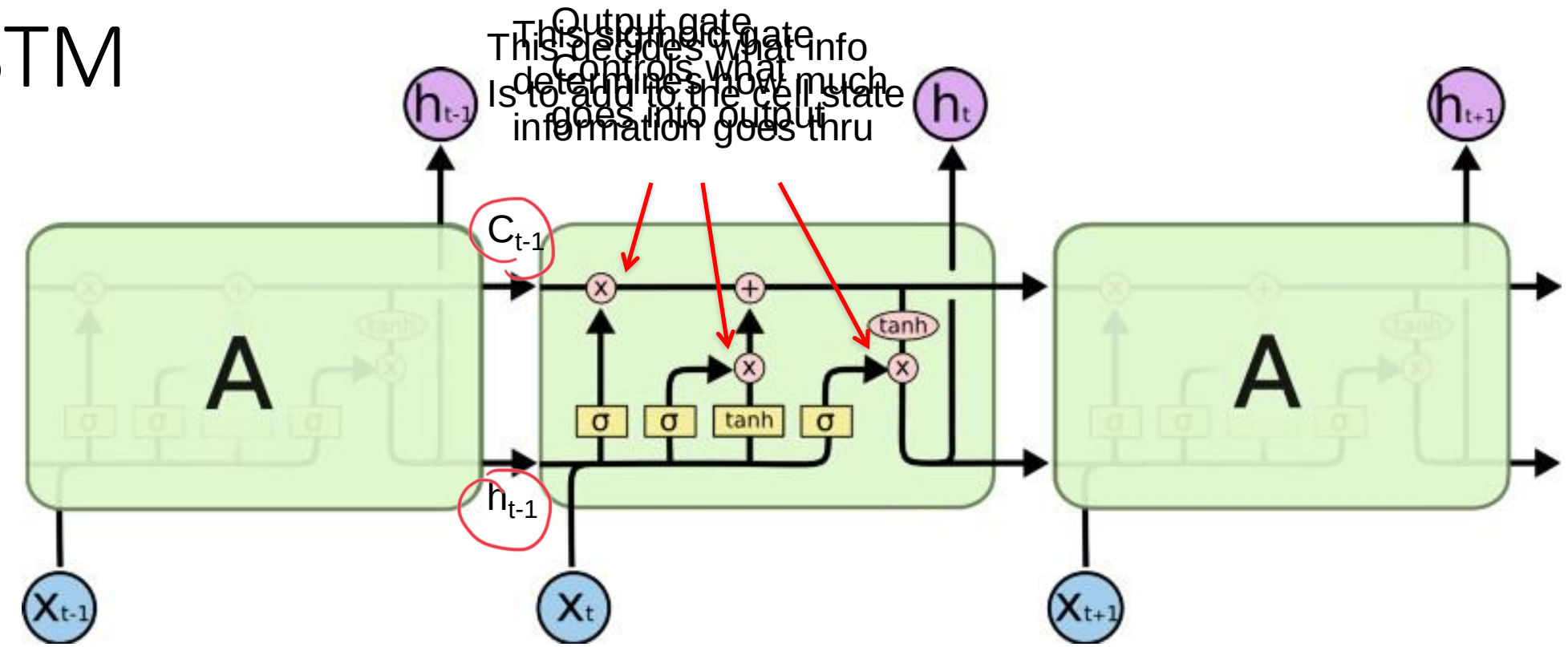
Sepp Hochreiter
Fakultät für Informatik, Technische Universität München, 80290 München, Germany

Jürgen Schmidhuber
IDSIA, Corso Elvezia 36, 6900 Lugano, Switzerland

Learning to store information over extended time intervals by recurrent backpropagation takes a very long time, mostly because of insufficient, decaying error backflow. We briefly review Hochreiter's (1991) analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called long short-term memory (LSTM). Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete-time steps by enforcing constant error flow through constant error carousels within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local in space and time; its computational complexity per time step and weight is $O(1)$. Our experiments with artificial data involve local, distributed, real-valued, and noisy pattern representations. In compar-

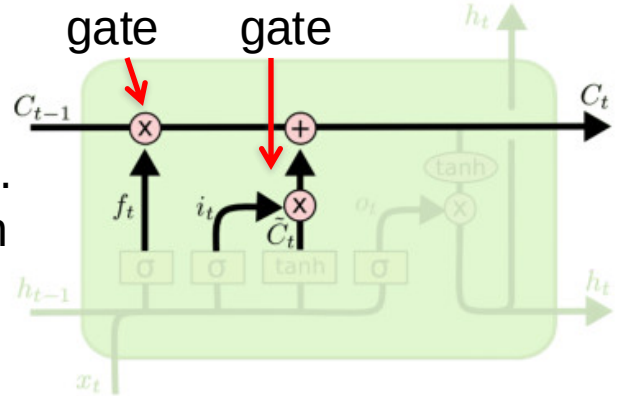


LSTM



Forget gate

input gate



The core idea is this cell state C_t, it is changed slowly, with only minor linear interactions. It is very easy for information to flow along it unchanged.

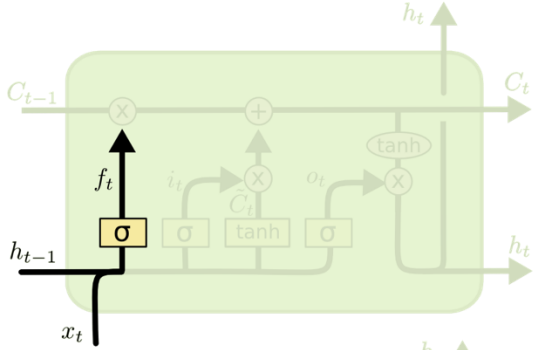
Why sigmoid or tanh:
Sigmoid: 0,1 gating as switch.
Vanishing gradient problem in LSTM is handled already.
ReLU replaces tanh ok?

forget gate.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

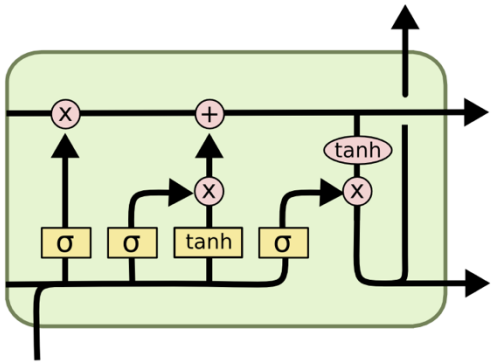
→ short mg.

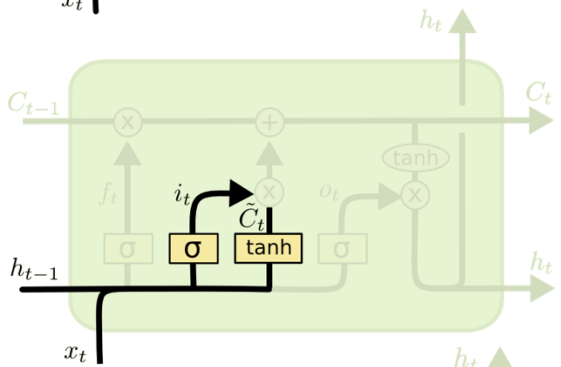
input dupe.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

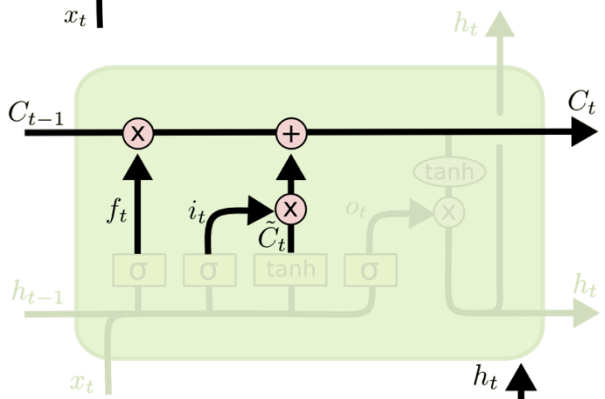
forget gate.





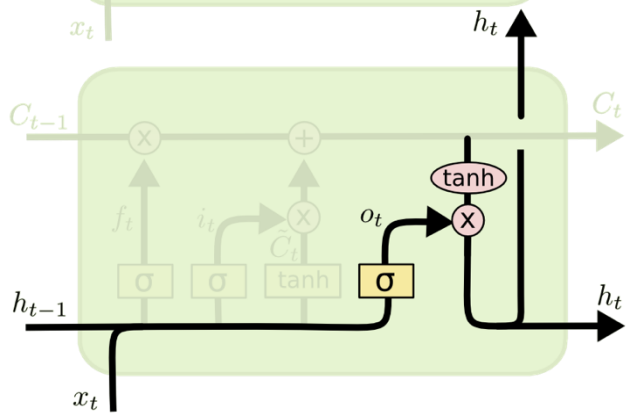
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

i_t decides what component is to be updated.
C'_t provides change contents



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Updating the cell state

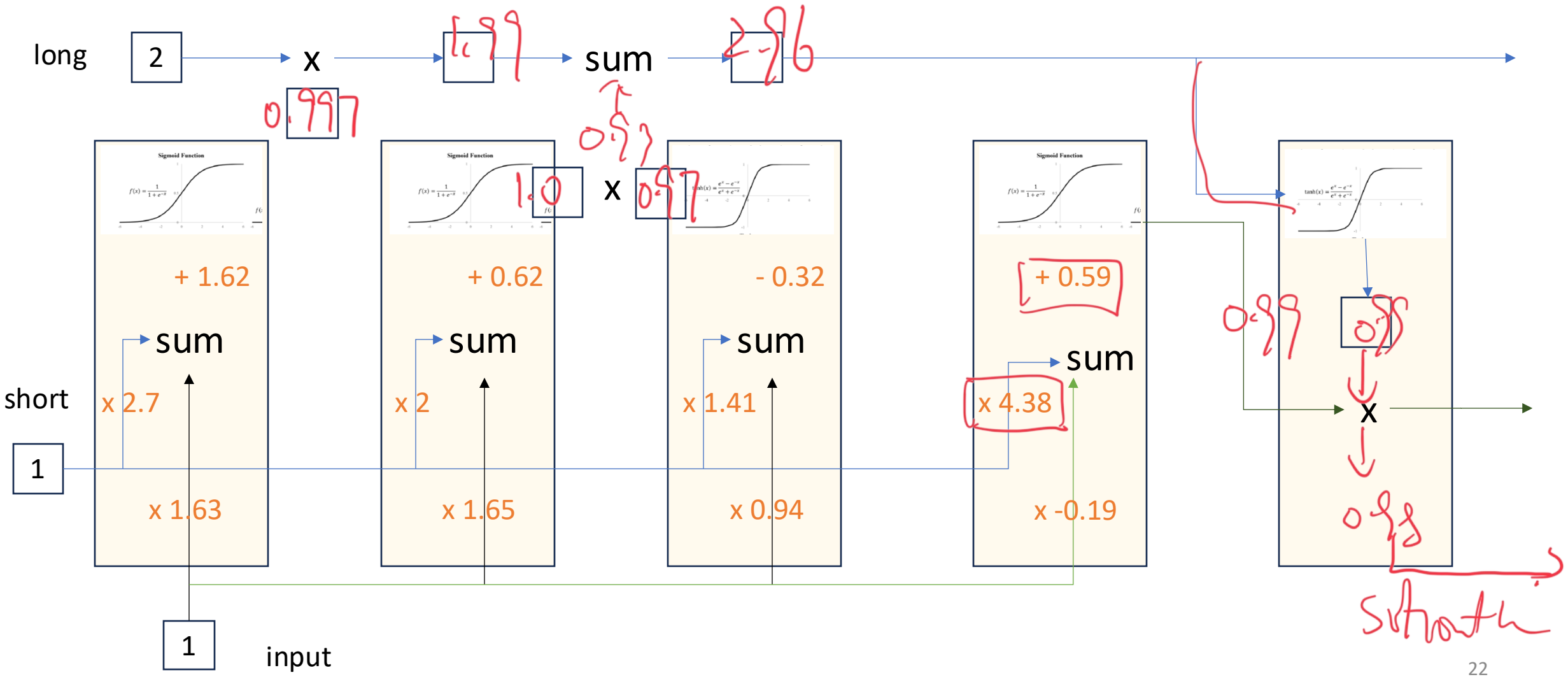


$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

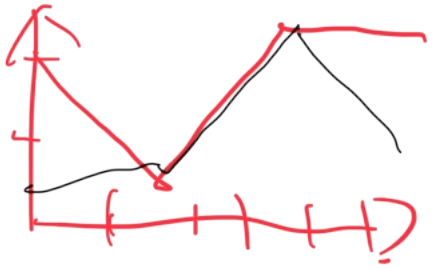
updat should turn my

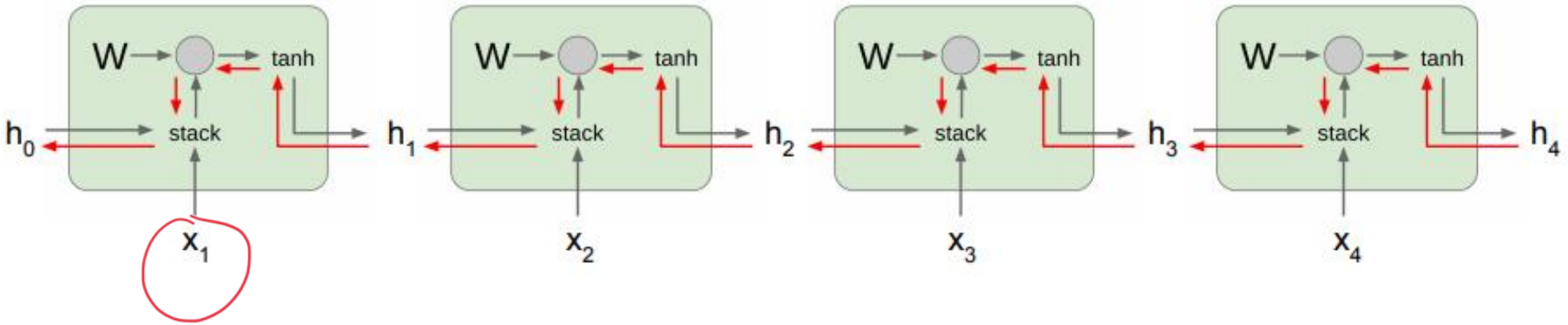
Decide what part of the cell state to output

Long Short-Term Memory (LSTM)



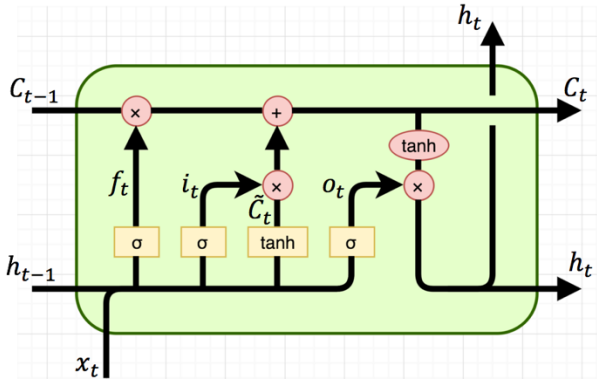
Long Short-Term Memory (LSTM)



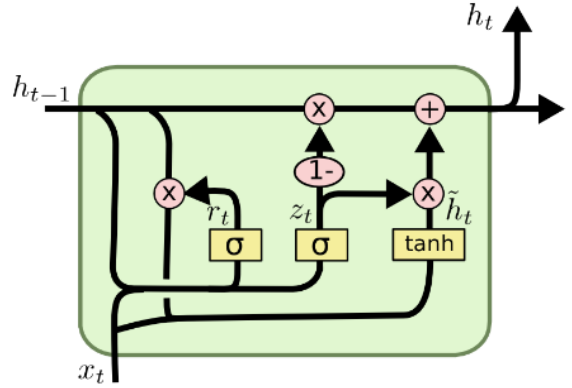


Long Short-Term Memory (LSTM)

- Vanilla RNN operates in a “multiplicative” way (repeated tanh).
- Two recurrent cell designs were proposed and widely adopted:
  - **Long Short-Term Memory (LSTM)** (Hochreiter and Schmidhuber, 1997)
  - Gated Recurrent Unit (GRU) (Cho et al. 2014)
- Both designs process information in an “additive” way with gates to control information flow.
  - **Sigmoid gate outputs numbers between 0 and 1, describing how much of each component should be let through.**



Standard LSTM Cell



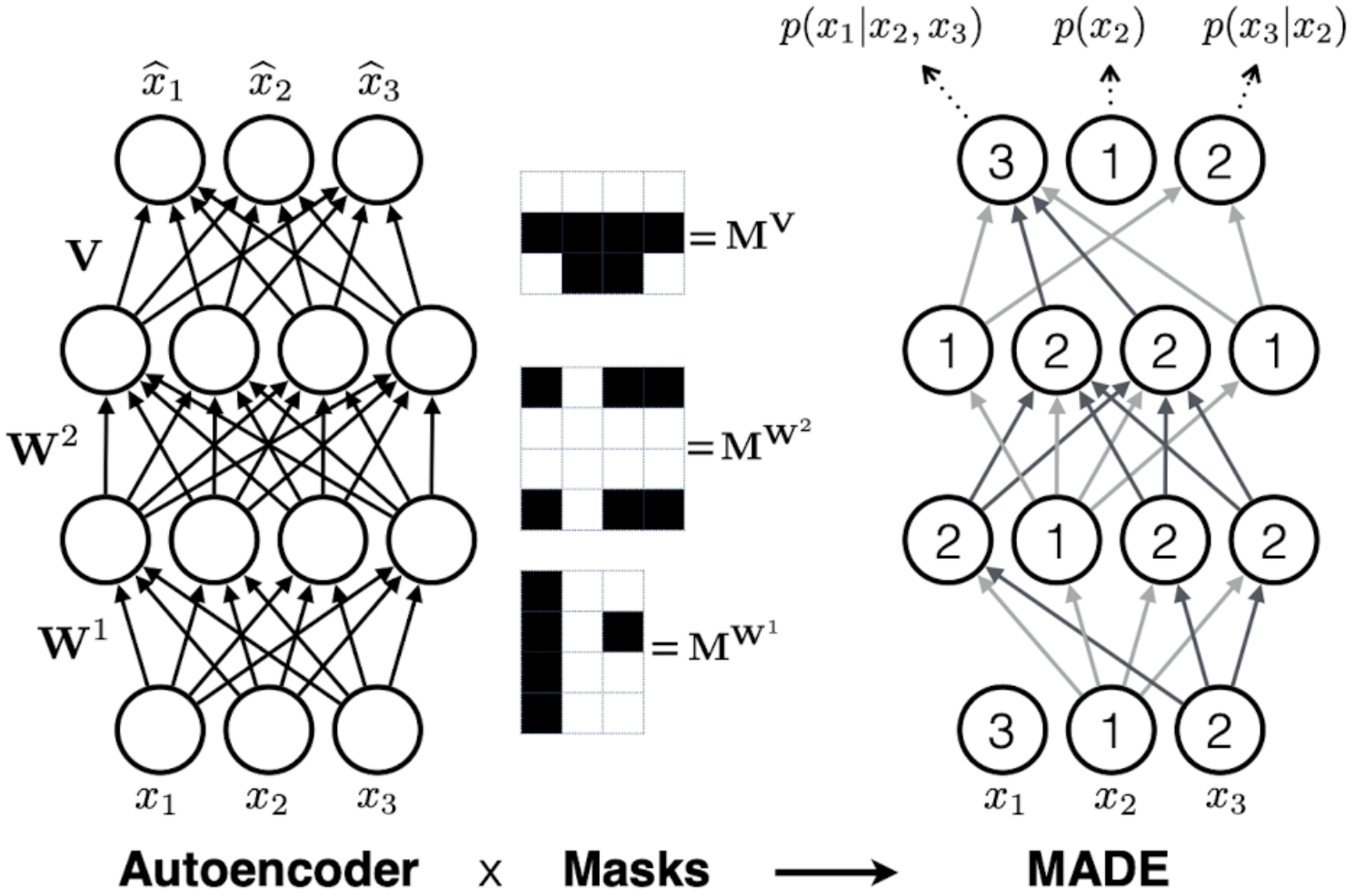
GRU Cell

Masking-based Autoregressive Models

Second major branch of neural AR models

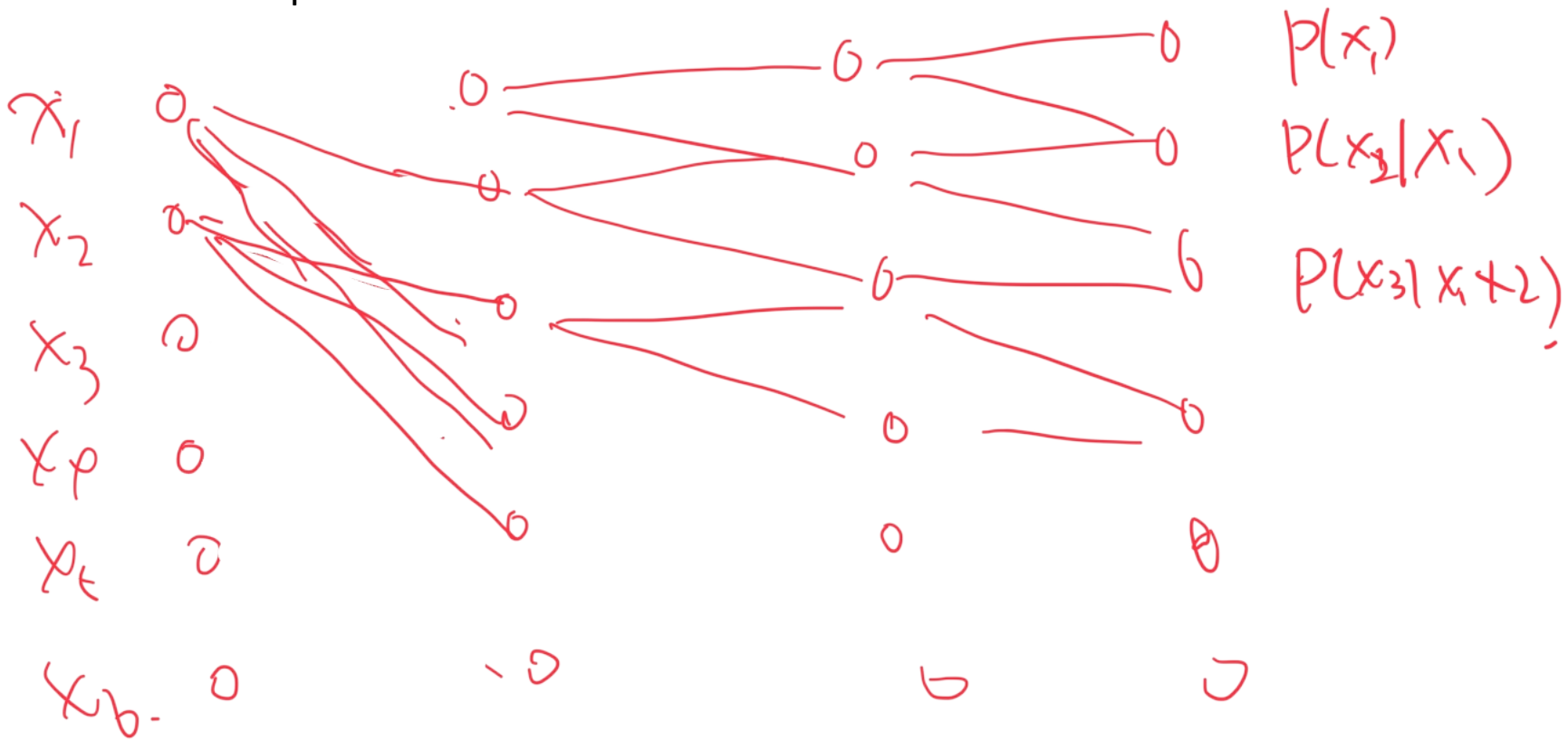
- **Key property:** parallelized computation of all conditions
- Masked MLP (MADE)
- Masked convolutions & self-attention (next lecture)
  - Also share parameters across time

Masked Autoencoder for Distribution Estimation (MADE)

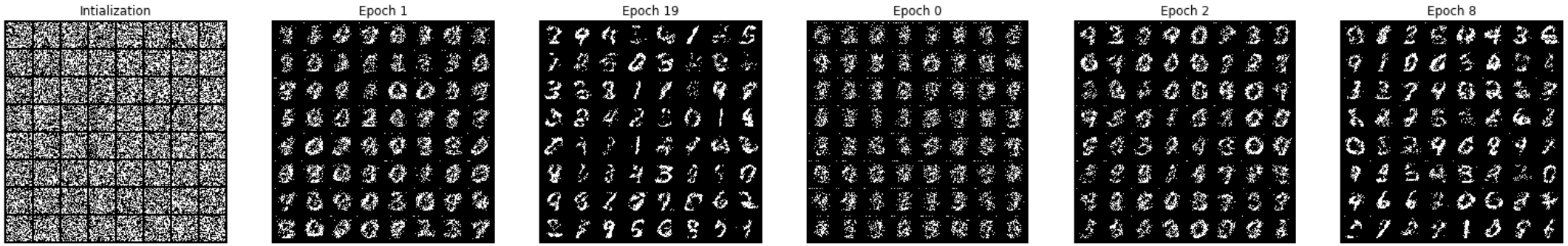


Masked Autoencoder for Distribution Estimation (MADE)

General Principle



Masked Autoencoder for Distribution Estimation (MADE)



Figures from CS 294 UC Berkeley

Masked Autoencoder for Distribution Estimation (MADE)

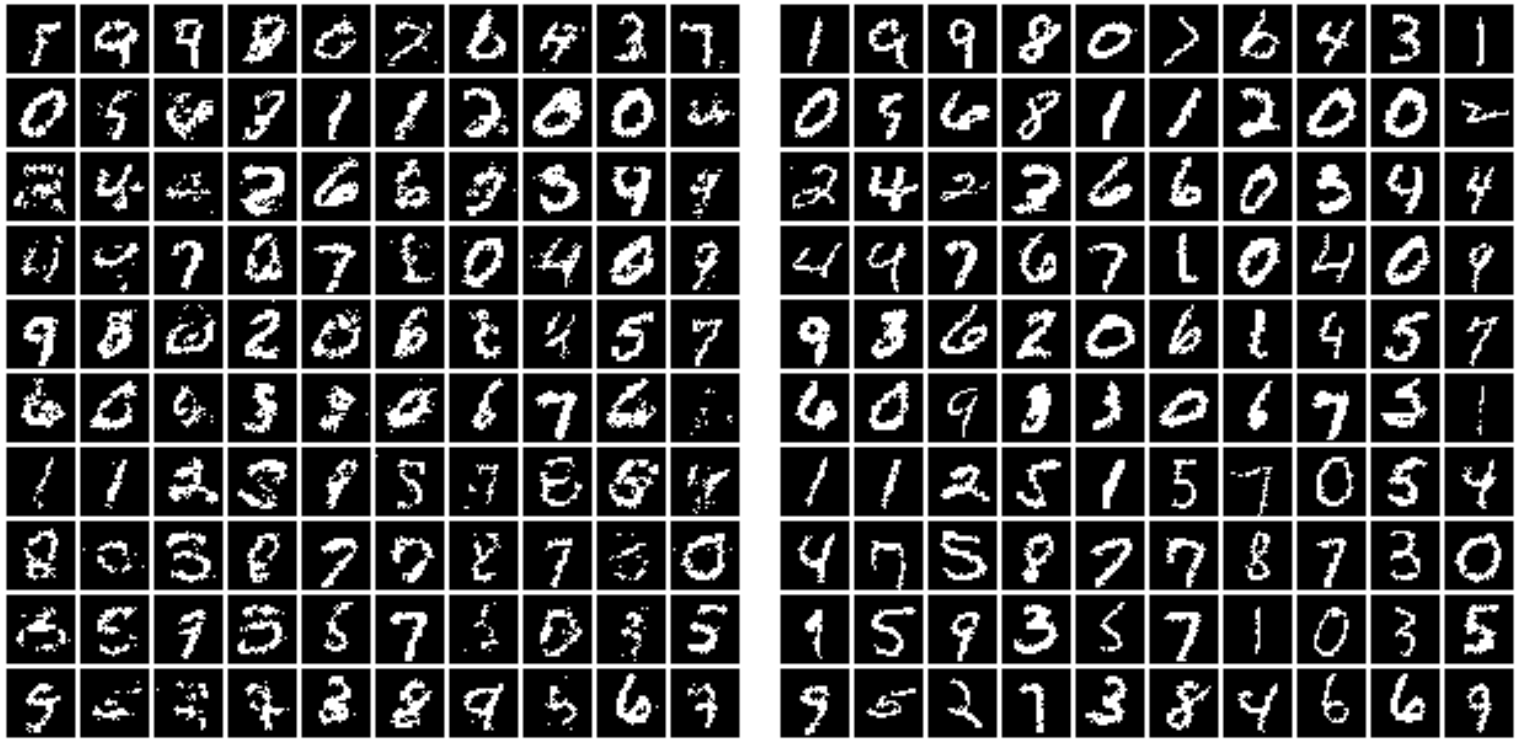


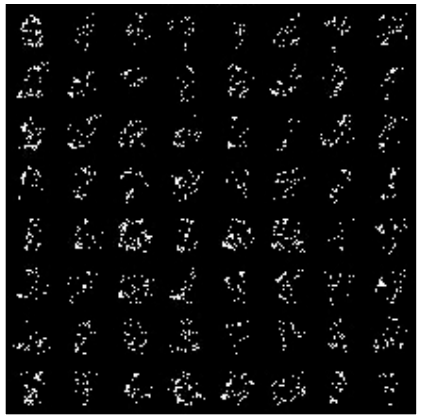
Figure 3. Left: Samples from a 2 hidden layer MADE. Right: Nearest neighbour in binarized MNIST.

Masked Autoencoder for Distribution Estimation (MADE)

All orderings achieve roughly the same bits per dim, but samples are different

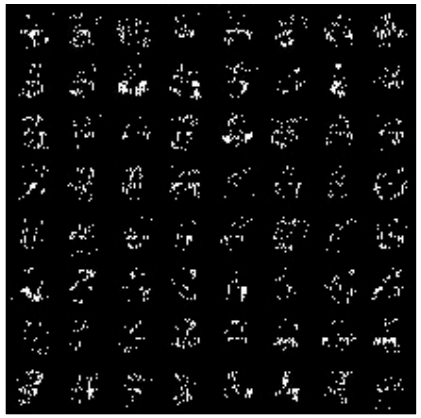
Random Permutation

Samples



Even then Odd Indices

Samples

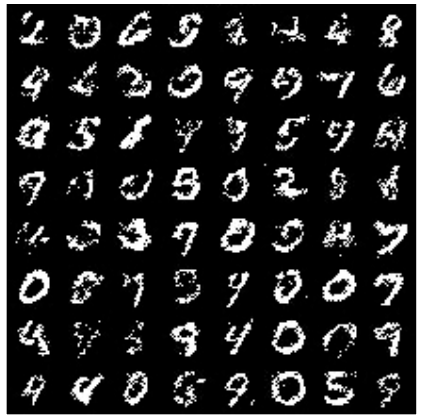


Rows (Raster Scan)

Samples

Columns

Samples

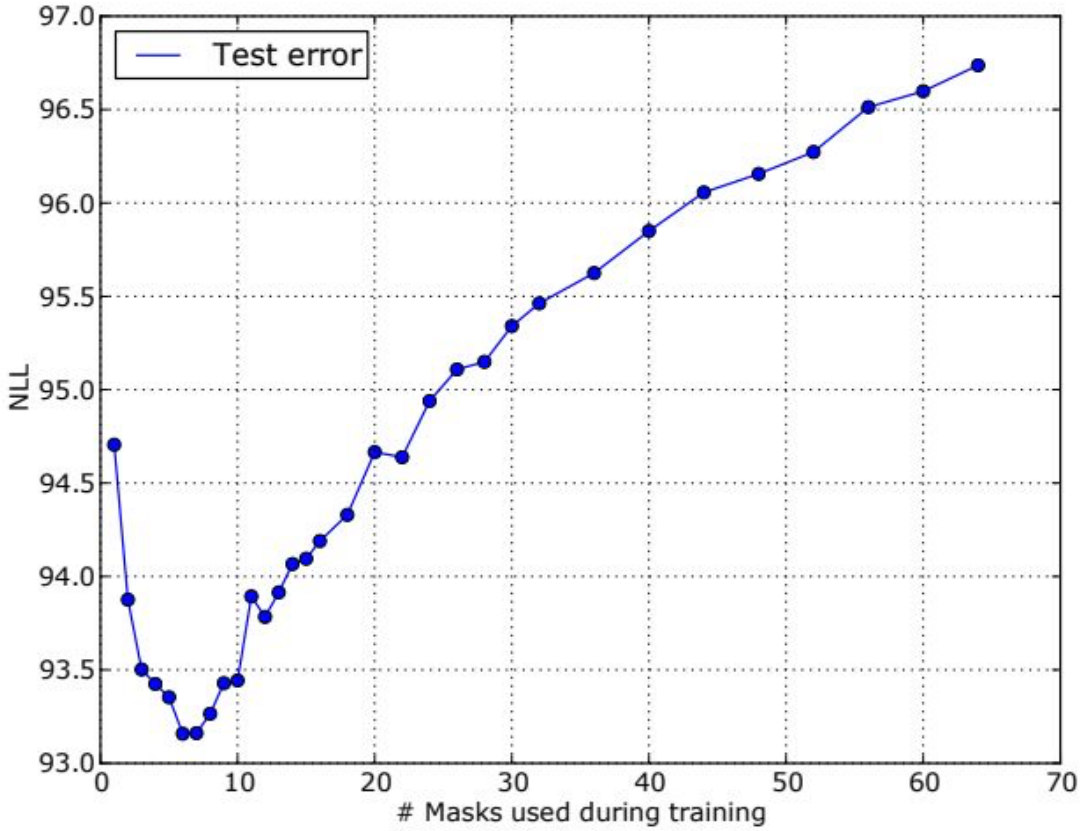


Top to Middle, Bottom to Middle

Samples

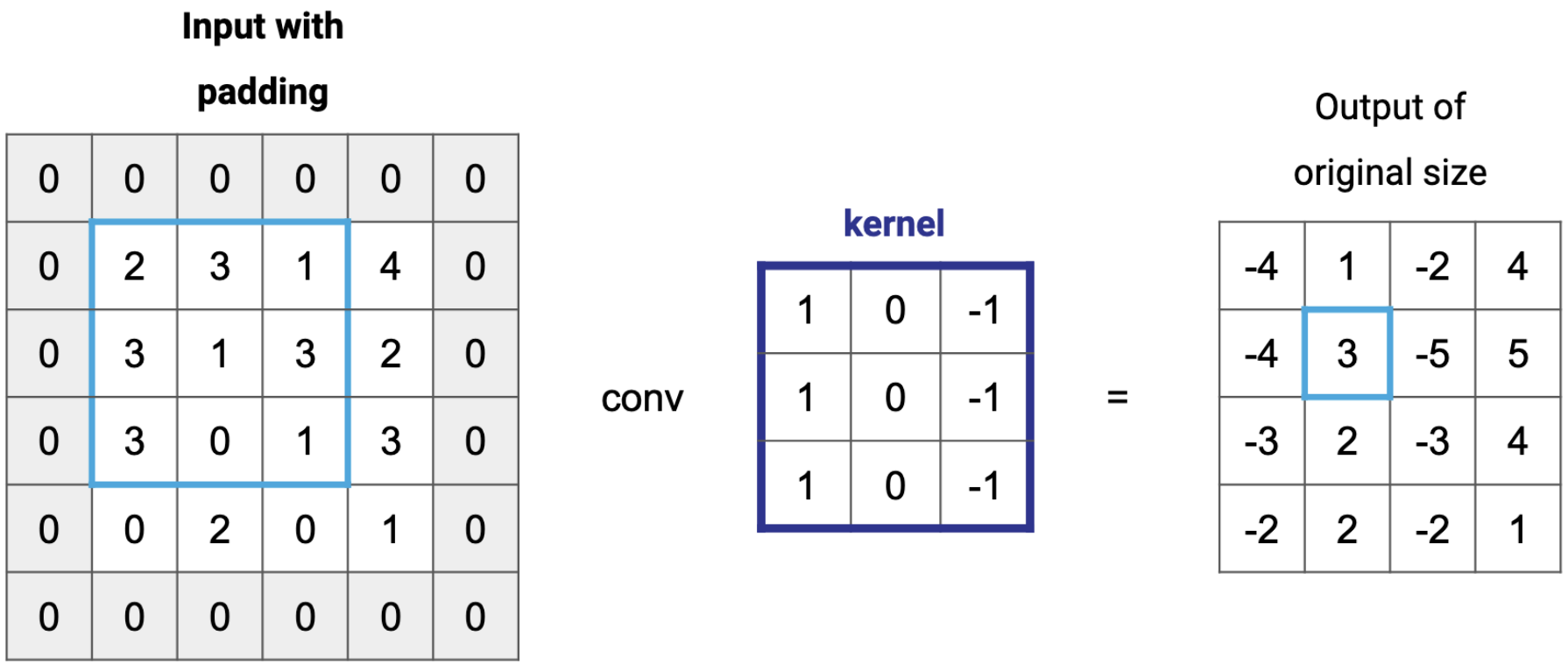
Figures from CS 294 UC Berkeley

Masked Autoencoder for Distribution Estimation (MADE)



Refresh: Convolution

- Convolution: slides a filter (kernel) over an input (e.g., image or other signals) to compute a weighted sum at each position.



Convolution: Translation Equivariance

- Good “inductive bias” for natural images

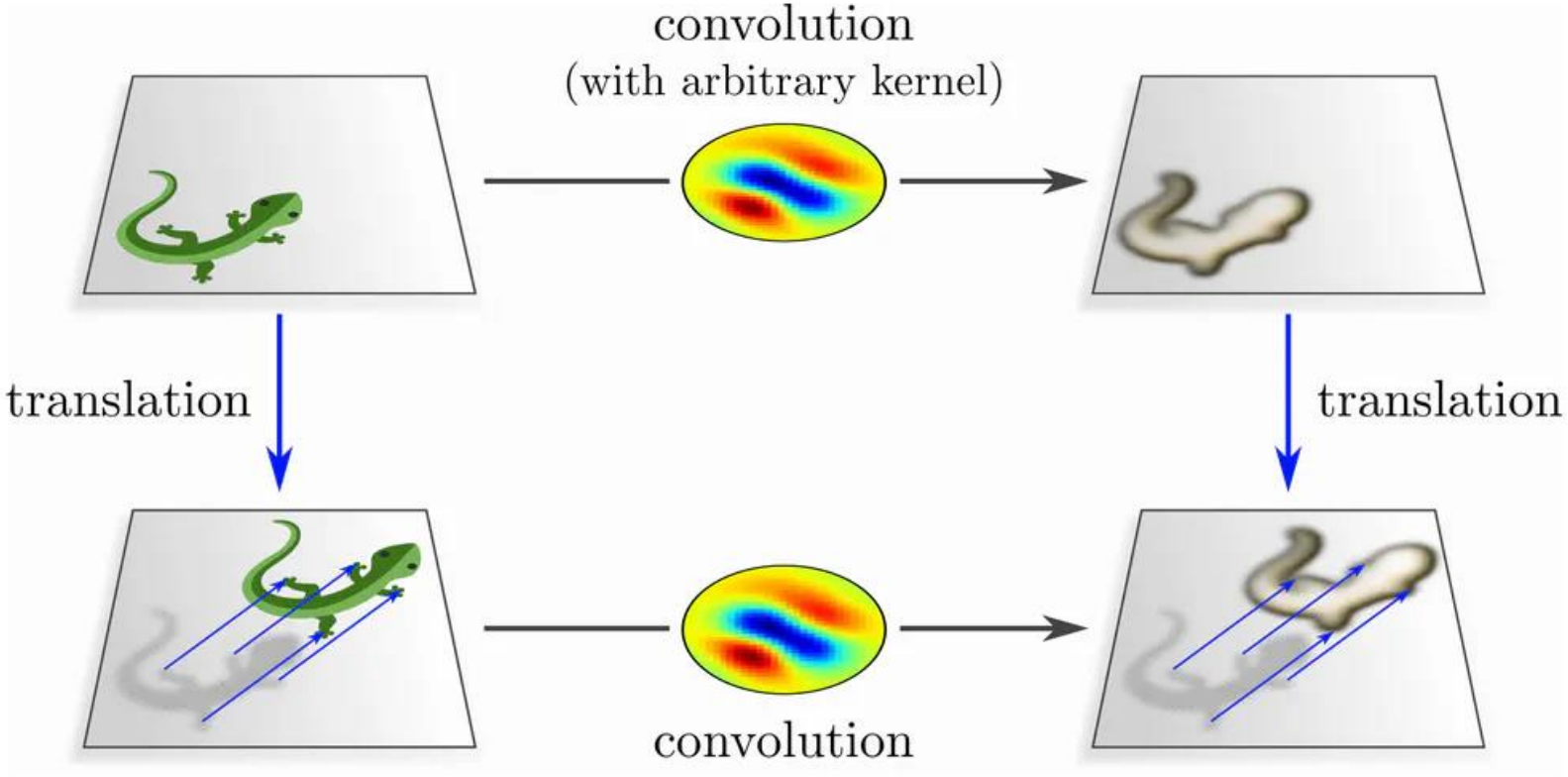
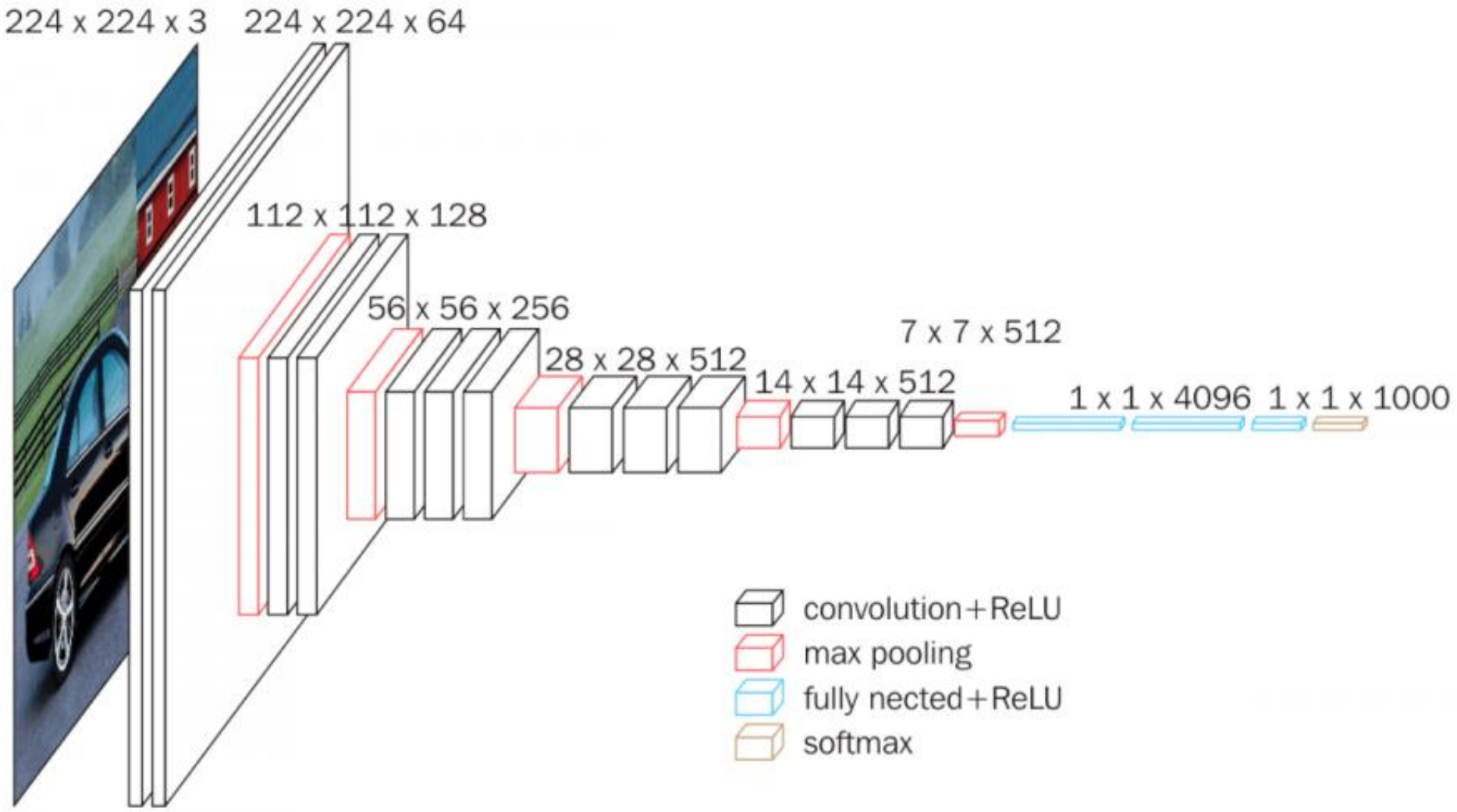


Figure from: https://maurice-weiler.gitlab.io/blog_post/cnn-book_2_conventional_cnns/

Convolutional Neural Networks (CNNs) for Image Classification



Receptive Field

- Receptive field: the size of the input region that affects a latent neuron
- Receptive field become bigger as we go deeper

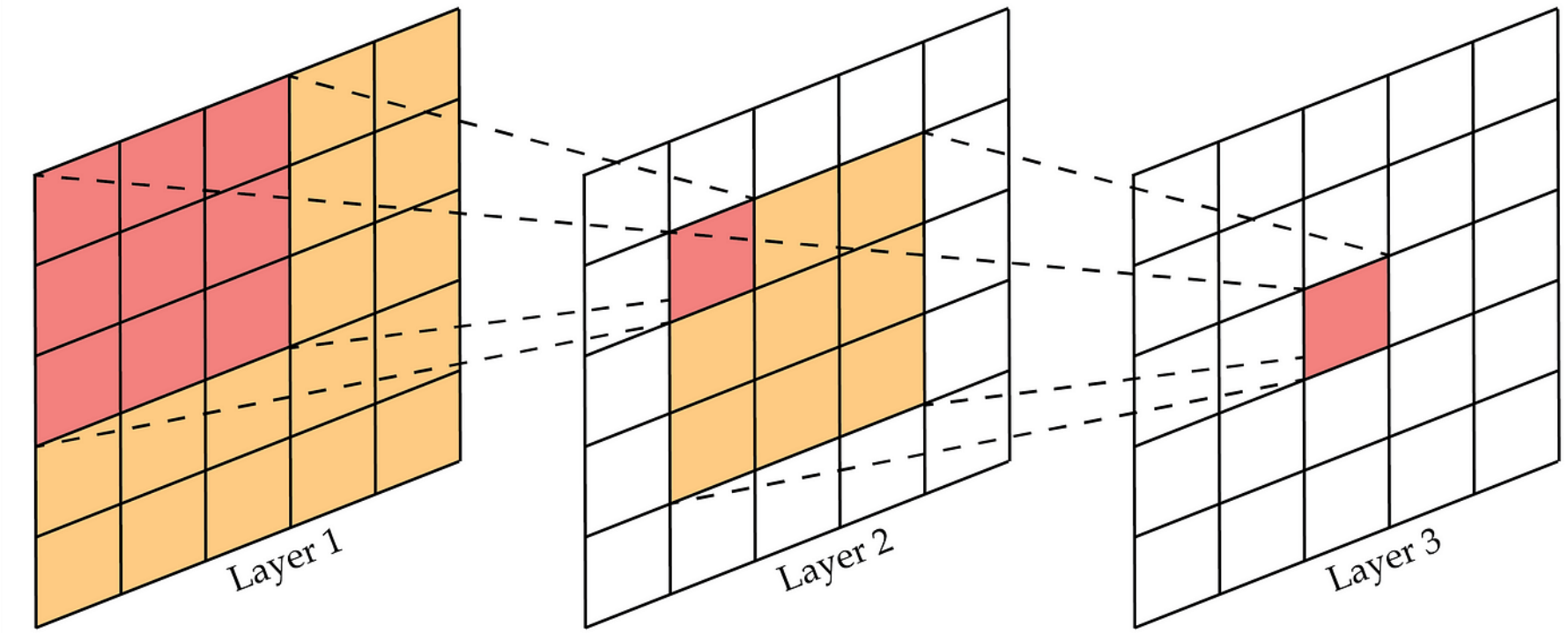
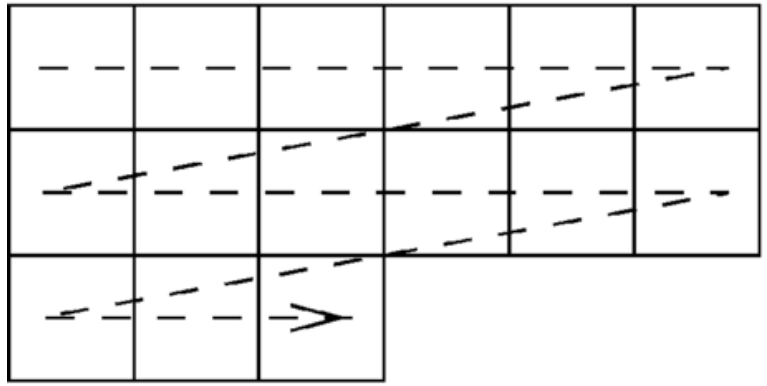


Figure from: https://medium.com/@rekalantar/receptive-fields-in-deep-convolutional-networks-43871d2ef2e9

PixelCNN: Convolution with Causal Masks

- We assume a “raster order” of pixels
- Convolution with “causal” mask



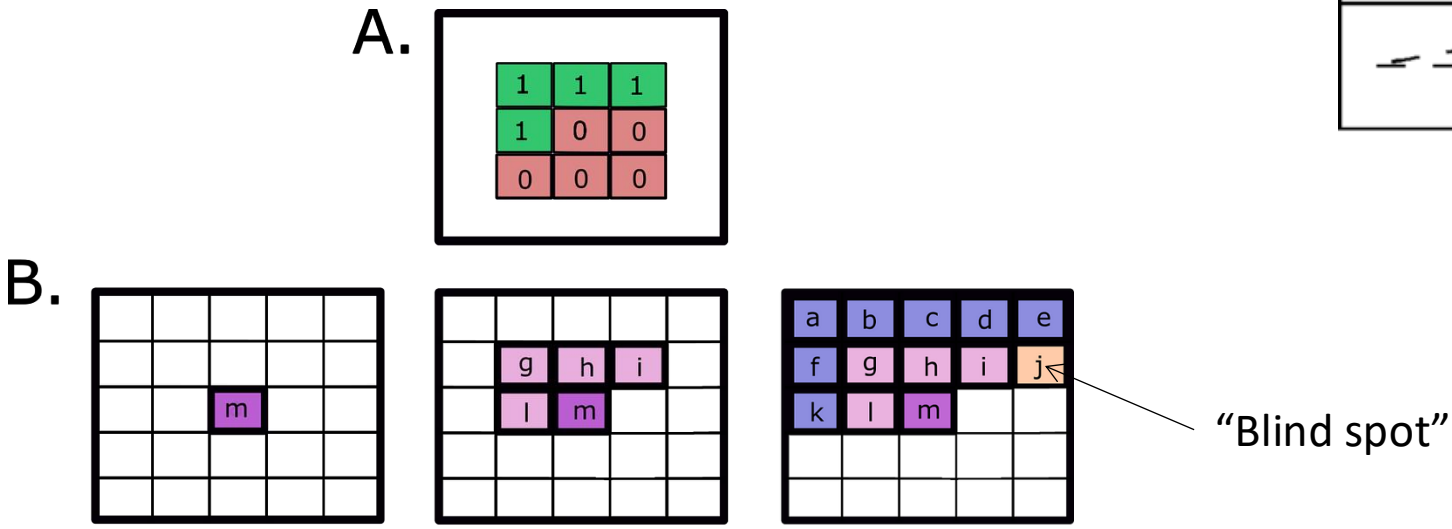
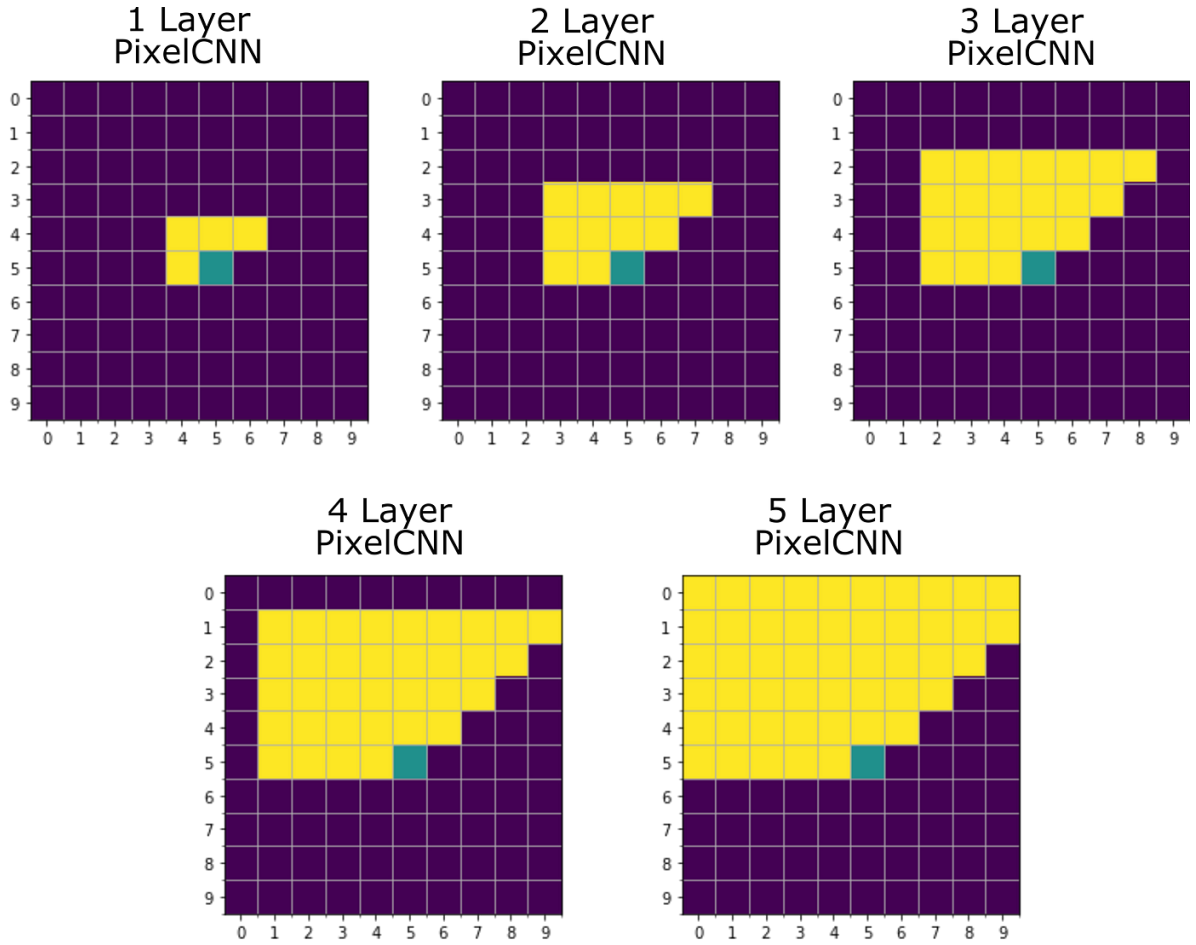


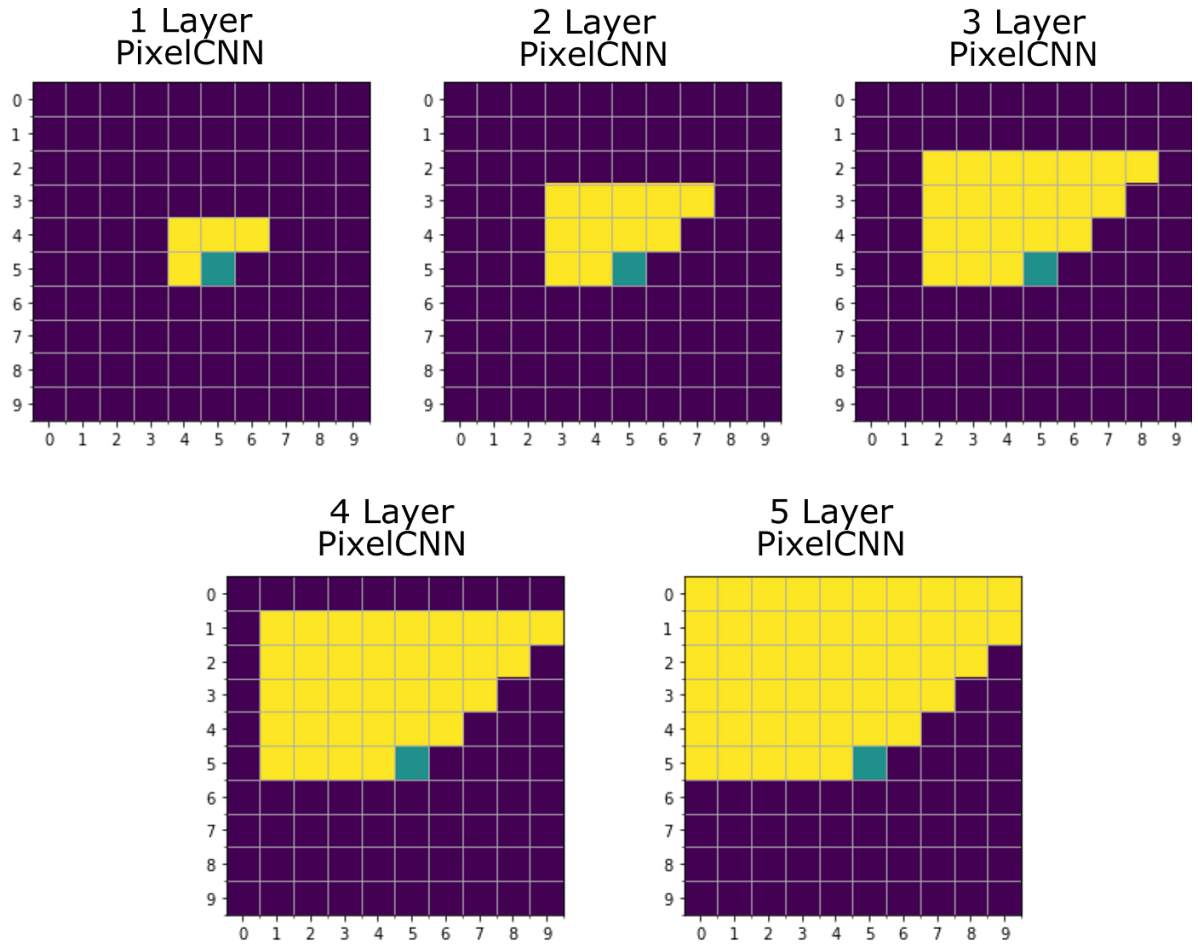
Figure from: https://towardsdatascience.com/pixelcnns-blind-spot-84e19a3797b9

PixelCNN: Convolution with Causal Masks



Receptive field after stacking multiple layers

PixelCNN: Convolution with Causal Masks



Receptive field after stacking multiple layers

Quiz

Password: Cat